**Image Classification on CIFAR-10 dataset.**

Objective:

To design a network combining supervised and unsupervised architectures in one model to achieve a classification task on Cifar-10 dataset under the condition that only 50% of the following classes (birds, deer, truck) can be used for training.

Solution:

1. Data Exploration.

This is performed for understanding the data.

There are 50000 images of shape 32 x32 x 3 flattened its pixels and stored as 50000 x 3072. Each class has 6000 images. The images can be visualised using plotting them.

1. Preparing training, testing and validation data.

Initially there are 50000 images for training and 10000 images for testing.

Considering the given condition, Only 50% of the images from birds, deer and trucks can be used for training. Ie, 2000 from each class in the training set will go to the test set. This will result in getting an unbalanced dataset.

After considering this, the size of the training set will be 44000 and the testing set will be 16000.

1. Preprocessing.

1. Converting the data type of training and testing numpy arrays to float32 format.

2. Normalizing

3. One hot encoding of labels.

1. Splitting 20% of training data as validation set.
2. Defining convolutional autoencoder for feature extraction.

A stack of conv2D and its pooling layer is defined.

CNN takes tensors of shape(image\_height, image\_width,num of color channels) .

Here, the model is built by two convolutional layers stacked up with its pooling layer giving the encoder base of stacked autoencoders.

1. Once the model is created, it has to be compiled using loss function and an optimizer.

The loss function used here is mean\_squared\_error and optimizer is RMSprop()

1. Now we can visualize the model using summary() which displays all the layers and number of parameters (bias and weights).
2. Inorder to deal with the imbalance in the classes, we use class\_weight from scikit-learn library. This will produce balanced probabilities for each class.
3. Training autoencoder

Models can be trained using fit function. We can pass the number of epochs (Here, epoch=50), training data, batch size (here, batch size=128) and validation set. This will return a history object which can be used for evaluation.

1. Use the trained model to predict the reconstructed image using test data.
2. Once verified, we can build a CNN for classification and use the middle layer of autoencoder as its input.

Hence the overall design goes like:



1. Building a convolutional network.

In this solution, the input to the CNN will be the images produced using modeling the features of images obtained by the autoencoder in its middle layer.

Basically, a CNN network is built using the following parameters.

1. Number of Filters,
2. Size of kernels for pooling.
3. Input image

This convolution layer + pooling layer constitutes one layer of CNN.

After convolution, the output is flattened to get the fully connected layer and there we apply softmax classifier for classification.

13. Once the model is built, it's time to compile it. For compiling, the following three parameters used:

1. Loss function [categorical\_crossentropy is used here]
2. Optimizer [adam optimizer is used here]
3. Metrics for evaluation [Accuracy is used here]

14. After compilation, the next step is to train this network using fit() and its parameters as:

1. images produced from the encoded features and its corresponding labels.
2. Batches
3. Epochs.
4. class weights

15. Once the model is trained it will be saved as a history object and can be used for evaluation.

RESULTS

Classification Report:

precision recall f1-score support

Airplane 0.63 0.83 0.71 1000

Automobile 0.75 0.88 0.81 1000

Bird 0.86 0.62 0.72 3000

Cat 0.50 0.55 0.53 1000

Deer 0.81 0.72 0.76 3000

Dog 0.52 0.71 0.60 1000

Frog 0.65 0.84 0.74 1000

Horse 0.61 0.81 0.70 1000

Ship 0.83 0.87 0.85 1000

Truck 0.95 0.80 0.87 3000

accuracy 0.75 16000

macro avg 0.71 0.76 0.73 16000

weighted avg 0.77 0.75 0.75 16000

